(51) International Patent Classification ^{6} :		(11) International Publication Number: WO 99/21091
G06F 11/14, 11/00	A1	(43) International Publication Date: 29 April 1999 (29.04.99)

(21) International Application Number: PCT/US98/22160

(22) International Filing Date: 20 October 1998 (20.10.98)

(30) Priority Data:
08/967,341   21 October 1997 (21.10.97)   US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: MOIIN, Hossein; 355 Crestmont Drive, San Francisco, CA 94131 (US). PRUSCINO, Angelo; Apartment A106, 550 Ortega Avenue, Mountain View, CA 94040 (US).

(74) Agents: HYMAN, Eric, S. et al.; Blakely, Sokoloff, Taylor & Zafman, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025–1026 (US).

(81) Designated States: AL, AM, AT, AT (Utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), EE, EE (Utility model), ES, FI, FI (Utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

(54) Title: DATA INTEGRITY AND AVAILABILITY IN A DISTRIBUTED COMPUTER SYSTEM

(57) Abstract

Data integrity and availability is assured by preventing a node of a distributed, clustered system from accessing shared data in the case of a failure of the node or communication links with the node. The node is prevented from accessing the shared data in the presence of such a failure by ensuring that such a failure is detected in less time than a secondary node would allow user I/O activities to commence after reconfiguration. The prompt detection of failure is assured by periodically determining which configuration of the current cluster each node believes itself to be a member of. Each node maintains a sequence number which identifies the current configuration of the cluster. Periodically, each node exchanges its sequence number with all other nodes of the cluster. If a particular node detects that it believes itself to be a member of a preceding configuration to that to which another node belongs, the node determines that the cluster has been reconfigured since the node last performed a reconfiguration. Therefore, the node must no longer be a member of the cluster. The node then refrains from accessing shared data. In addition, if a node suspects a failure in the cluster, the node broadcasts a reconfigure message to all other nodes of the cluster through a public network. Since the messages are sent through a public network, failure of the private communications links between the nodes does not prevent receipt of the reconfigure messages.

DATA INTEGRITY AND AVAILABILITY IN A DISTRIBUTED COMPUTER SYSTEM

SPECIFICATION

FIELD OF THE INVENTION

The present invention relates to fault tolerance in distributed computer systems and, in particular, to a particularly robust mechanism for assuring integrity and availability of data in the event of one or more failures of nodes and/or resources of a distributed system.

BACKGROUND OF THE INVENTION

1. Introduction and Background

Main motivations for distributed computer systems are the addition of high availability and increased performance. Distributed computer systems support a host of highly-available or parallel applications, such as Oracle Parallel Server (OPS) Informix XPS or HA-NFS. While the key to the success of distributed computer systems in the high-end market has been high-availability and scalable performance, another key has been the implicit guarantee that the data trusted to such distributed computer system will remain integral.

2. System Model and Classes of Failures

The informal system model for some distributed computer systems is that of a "trusted" asynchronous distributed system. The system is composed of 2 to 4 nodes that communicate via message exchange on a private communication network. Each node of the systems has two paths into the communication medium so that the failure of one path does not isolate the node from other cluster members. The system has a notion of membership that guarantees that the nodes come to a consistent agreement on the set of member nodes at any given time. The system is capable of tolerating failures and the failed nodes are guaranteed to be removed from the cluster within bounded time by using fail-fast drivers and timeouts. The nodes of the distributed system are also connected to an external and public network that connects the client machines to the cluster. The storage subsystem may

contain shared data that may be accessed from different nodes of the cluster simultaneously. The simultaneous access to the data is governed by the upper layer applications. Specifically, if the application is OPS then simultaneous access is granted as OPS assumes that the underlying architecture of the system is a shared-disk architecture. All other applications that are supported on the cluster assume a shared nothing architecture and therefore, do not require simultaneous access to the data.

While the members of a cluster are "trusted", nodes that are not part of the current membership set are considered "un-trusted". The un-trusted nodes should not be able to access shared resources of the cluster. These shared resources are the storage subsystem and the communication medium. While the access to the communication medium may not pose a great threat to the operation of the cluster, other than a possible flooding of the medium by the offending node(s), the access to the shared storage sub-system constitutes a serious danger to the integrity of the system as an un-trusted node may corrupt the shared data and compromise the underlying database. To fence non-member nodes from access to the storage sub-system the nodes that are members of the cluster *exclusively* reserve the parts of the storage sub-system that they "own". This results in exclusion of *all* other nodes, regardless of their membership status, from accessing the fenced parts of the storage sub-system. The fencing has been done, up to now, via low level SCSI-2 reservation techniques, but it is possible to fence the shared data by the optional SCSI-3 persistent group reservations as those are, in fact, a super-set of SCSI-2 reservations. It is important to note that we assume that the nodes that could possibly form the cluster, regardless of whether they are current cluster members or not, do not behave maliciously. The cluster does not employ any mechanisms, other than requiring root privileges for the user, to prevent malicious adversaries from gaining membership in the cluster and corrupting the shared database.

While it is easy to fence a shared storage device if that device is dual-ported, via the SCSI-2 reservations, no such technique is available for multi-ported devices, as the necessary but optional SCSI-3 reservations are not implemented by the disk drive vendors. In this paper we assume that the storage sub-system is entirely composed of either dual-ported or multi-ported devices. Mixing the dual and multi-ported storage devices does not add any complexity to our algorithms, but will make the discussion more difficult to follow. It should be pointed out, however, that a multi-ported storage sub-system has better

availability characteristics than a dual-ported one as more node failures can be tolerated in the system without loss of access to the data. Before we investigate the issues of availability, data integrity, and performance we should classify the nature of faults that our systems are expected to tolerate.

There are many possible ways of classifying the types of failures that may occur in a system. The following classification is based on the *intent* of the faulty party. The intent defines the nature of the faults and can lead the system designer to guard against the consequences of such failures. Three classes of failures can be defined on the basis of the intent:

1.   **No-Fault Failures**: This class of failures include various hardware and software failures of the system such as node failures, the communication medium failures, or the storage sub-system failures. All these failures share the characteristic that they are not the result of any misbehavior on the part of the user or the operator. A highly-available system is expected to tolerate some such failures. The degree to which a system is capable of tolerating such failures and the affect on the users of the system determine the class (e.g, fault-tolerant or highly-available) and level (how many and what type of failures can be tolerated simultaneously or consecutively) of availability in a traditional paradigm.

2.   **Inadvertent Failures**: The typical failure in this class is that of an operator mistake or pilot error. The user that causes a failure in this class does not intend to damage the system, however, he or she is relied upon to make the right decisions and deviations from those decisions can cause significant damage to the system and its data. The amount of protection the system incorporates against such failures defines the level of trust that exists between the system and its users and operators. A typical example of this trust in a UNIX environment is that of a user with root privileges that is relied upon to behave responsibly and not delete or modify files owned and used by other users. Some distributed systems assume the same level of trust as the operating system and restricts all the activities that can affect other nodes or users and their data to a user with root privileges.

3.   **Malicious Failures**: This is the most difficult class of failures to guard against and is generally solved by use of authenticated signatures or similar security

techniques. Most systems that are vulnerable to attacks by malicious users must take extra security measures to prevent access to such users. Clusters, in general, and Sun Cluster 2.0 available from Sun Microsystems, Inc. of Palo Alto, California, in particular, are typically used as back-end systems and are assumed immune from malicious attacks as they are not directly visible to users outside the local area network in which they are connected to a selected number of clients. As an example of this lack of security, consider a user that can break into a node and then joins that node as a member of cluster of a distributed system. The malicious user can now corrupt the database by writing on the shared data and not following the appropriate protocols, such as acquiring the necessary data locks. This lack of security software is due to the fact that some distributed systems are generally assumed to operate in a "trusted" environment. Furthermore, such systems are used as high-performance data servers that cannot tolerate the extra cost of running security software required to defend the integrity of the system from attack by malicious users.

Note that the above classification is neither comprehensive nor that the classes are distinct. However, such classification serves as a model for discussing the possible failures in a distributed system. As mentioned earlier, we must guard against no-fault failures and make it difficult for inadvertent failures to occur. We do not plan to incorporate any techniques in system software to reduce the probability of malicious users from gaining access to the system. Instead, we can offer third party solutions that disallow access to potentially malicious parties. One such solution is the Fire Wall-1 product by Check Point Software Technologies Limited which controls access, connection, and provides for authentication. Note that the addition of security to a cluster greatly increases the cost and complexity of communication among nodes and significantly reduces the performance of that system. Due to the performance requirements of high-end systems, such systems typically incorporate security checks in the software layer that interacts directly with the public network and assume that the member nodes are trusted so that the distributed protocols, such as membership, do not need to embed security in their designs.

SUMMARY OF THE INVENTION

In accordance with the present invention, data integrity and availability is assured by preventing a node of a distributed, clustered system from accessing shared data in the case of a failure of the node or communication links with the node. The node is prevented from accessing the shared data in the presence of such a failure by ensuring that such a failure is detected in less time than a secondary node would allow user I/O activities to commence after reconfiguration.

The prompt detection of failure is assured by periodically determining which configuration of the current cluster each node believes itself to be a member of. Each node maintains a sequence number which identifies the current configuration of the cluster. Periodically, each node exchanges its sequence number with all other nodes of the cluster. If a particular node detects that it believes itself to be a member of a preceding configuration to that to which another node belongs, the node determines that the cluster has been reconfigured since the node last performed a reconfiguration. Therefore, the node must no longer be a member of the cluster. The node then refrains from accessing shared data.

In addition, if a node suspects a failure in the cluster, the node broadcasts a reconfigure message to all other nodes of the cluster through a public network. Since the messages are sent through a public network, failure of the private communications links between the nodes does not prevent receipt of the reconfigure messages.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a distributed computer system which includes dual-ported devices.

Figure 2 is a block diagram of a distributed computer system which includes a multi-ported device.

Figure 3 is a logic flow diagram of the determination that a node has failed in accordance with the present invention.

Figure 4 is a logic flow diagram of the broadcasting of a reconfiguration message in accordance with the present invention.

DETAILED DESCRIPTION

3 Dual-Ported Architectures

A system with dual-ported storage sub-system is shown in Figure 1. In this figure, the storage device 111 is accessible from nodes 102A and 102D, storage device 112 is accessible from nodes 102A and 102B, storage device 113 from 102B and 102C, and storage device 114 from 102C and 102D. Note that in this configuration if the two nodes that are connected to a storage device fail, then the data on that storage device is no longer accessible. While the use of mirrored data can overcome this limitation in availability, the required mirroring software has the inherent limitation that it reduces the maximum performance of the system. Furthermore, note that not all the data is local to every node. This implies that a software solution is needed to allow remote access to the devices that are not local. In one embodiment, Netdisk provides such a function. Such a software solution requires the use of the communication medium and is inherently inferior in performance to a solution based on a multi-ported architecture. We will discuss the issues of performance and data availability a bit more below. For the remainder of this section we will concentrate on the issue of data integrity.

A system with dual-ported storage sub-system is shown in Figure 1. In a traditional cluster, data integrity in a system like that of Figure 1 is protected at three levels. First, we employ a robust membership algorithm that guarantees a single primary group will continue to function in the case the communication medium fails. Second, we use fail-fast drivers to guarantee that nodes that are "too slow" will not cause problems by not being able to follow the distributed protocols at the appropriate rate. Third, and as a last resort, we use the low level SCSI-2 reservations to fence nodes that are no longer part of the cluster from accessing the shared database. As the issue with multi-ported devices is that exclusive reservations such as those of SCSI-2 standard do not satisfy the requirements of the system, let us concentrate on the issue of disk-fencing.

Disk-fencing is used as a last resort to disallow nodes that are not members of the cluster from accessing the shared database. It operates by an IOCTL reserve command that grants exclusive access to the node requesting the reservation. The question to ask is what classes of failures does this disk fencing protect the system from? Clearly, this low level fencing does not protect the system from truly malicious users. Such adversaries can gain membership into the cluster and corrupt the database while the system's guards are down.

Some less intelligent adversaries who cannot gain membership, e.g., because they may not be able to obtain root's password, may be protected against if they happen to get into a node that is not a member. However, if an adversary can gain access to a non-member node of the cluster, could he or she not, and just as easily, gain access to a member node of the cluster? The final analysis regarding malicious adversaries, regardless of their level of sophistication, is that we do not protect against them and need to assume that the system is free of them.

The second category of failures defined above is inadvertent failures. Does disk-fencing provide any protection from inadvertent failures? Once again, the answer is "partially." While nodes that are not members of the clusters cannot write to (or read from) the shared database, those that are members can do so and the system relies on the users with the appropriate privileges to not interfere with the database and corrupt it. For instance, if the operator is logged into a node that is a cluster member and starts to delete files that are used by the database, or worse, starts to edit them, the system provides no protection against such interventions. While the user's intent is not that of a malicious adversary the effect and mechanisms he uses are the same and therefore, we cannot effectively guard against such inadvertent failures.

The third category of failures defined in the previous section is no-fault failures. A system, whether a clustered system or a single node system, should not allow such failures to result in data corruption. The problem is a trivial one for a single node system. If the system goes down, then there is no chance that the data will be corrupted. While obvious, this has an implication for the clustered systems; nodes that are down do not endanger the integrity of the database. Therefore, the only issue that can cause a problem with the data integrity in the system is that the nodes cannot communicate with each other and synchronize their access to the shared database. Let's look at an example of how such a failure can lead to data corruption and investigate the root cause of such a corruption.

Assume that the system of Figure 1 is used to run an OPS application. This means that the access to the shared database is controlled by the Distributed Lock Manager (DLM). Now assume that node *102A* is unable to communicate with the rest of the cluster. According to some membership algorithms, this failure will be detected and the remaining nodes of the system; nodes *102B*, *102C*, and *102D* will form a cluster that will not include node *102A*. If node *102A* does not detect its own failure in a timely manner, then that node

will assume that it is still mastering the locks that were trusted to it. As part of the cluster reconfiguration, other nodes will also think that they are mastering the locks previously owned by node *102A*. Now both sides of this cluster, i.e., the side consisting of node *102A* and the side with the membership set $\mu = \{102B, 102C, 102D\}$, assume that they can read from and write to the data blocks that they are currently mastering. This may result in data corruption if both sides decide to write to the same data block with different values. This potential data integrity problem is the reason why disk-fencing is used. Disk-fencing does cut the access of the nodes that are not part of the current membership to the shared devices by reserving such devices exclusively. In our example node *102B* would have reserved all the disks in storage device 112 and node *102D* would have done the same for all the disks in storage device 114. These reservations would have effectively cut off node *102A* off from the shared database and protect the integrity of the data.

It is interesting to note that while the above example deals with OPS that allows simultaneous access to the shared devices, the same arguments can be used for other applications that do not allow simultaneous access to the shared data. This observation is based on the fact that the underlying reason for data corruption is not the nature of the application but the inability of the system to detect the failure of the communication medium in a timely manner and therefore, allowing two primary components to operate on the same database. For all applications that need to take over the data accessible from a node that is no longer part of the membership set a similar reservation scheme is needed and the current software does provide such a mechanism.

While the use of disk-fencing is correct in principle, a closer look at the software used to implement the implicit guarantee of data integrity reveals some hidden "windows" that can violate such a guarantee. Furthermore, under most common operating conditions the low level reservations do not add any value to the system as far as data integrity is concerned. To understand the nature and extent of the "windows" that would violate the implicit guarantee of data integrity we must look at the actual implementation of disk-fencing. In some conventional systems, user I/O is indeed allowed prior to the commencement of disk-fencing. This clearly is a flaw that has been over-looked in such systems and allows for the possibility of multiple masters for a lock for a period that could last up to several minutes. This flaw in data integrity assurance is specific to application that use the CVM Cluster Volume Manager. This flaw, however, also points out to a

second phenomenon. Despite significant usage for a significant period of time of such
distributed systems, there have been no indications, as measured by customer complaints or
number of filed bug reports, that data integrity has been compromised.

The lack of complaints regarding data integrity clearly demonstrates that our three
tier protection scheme to ensure data integrity is in fact more than adequate and that the
first two layers of protection seem to suffice for all practical cases. A closer look at the
timing of events in the system and comparing the maximum duration of time required to
detect the failure of the communication medium, T_D, with the minimum amount of time
required to go through the steps of reconfiguration before user I/O is allowed, T_R, show
that in all practical cases, i.e., those cases where there is some amount of data to protect,
the membership algorithm and the fail-fast driver implementation are adequate means of
guaranteeing the integrity of the database. In general, if we can guarantee that $T_D < T_R$,
then we can guarantee that the membership algorithm will ensure the data integrity of the
shared database. It should also be pointed out that many commercial companies who are in
the business of fault-tolerant systems, e.g., Tandem Computers Incorporated or I.B.M., do
not provide the low-level protection of disk reservation for ensuring that the data trusted to
them remains integral. Therefore, what really is the key to providing true data integrity
guarantees to the users of a cluster is not the low level reservations, but the timely
detection of failures, and in particular, the timely detection of failures in the communication
medium. Disk-fencing although not strictly necessary is an extra security measure that
eases the customer's mind and is a strong selling point in commercial products. Below, we
will propose techniques that guarantee timely detection of communication medium failures.
While these techniques are proposed for a multi-ported architecture, they are equally
applicable to, although perhaps not as necessary for, dual-ported systems.

4 Proposed Multi-Ported Architectures

The system shown in Figure 2 is a minimal multi-ported architecture. In such a
system all the storage devices are connected to all the nodes of the cluster. For example, in
Figure 2 the storage device 210 is connected to all the nodes, *202A, 202B, 202C* and,
202D. In a typical system there may be many more devices such as storage device 210 that
are connected to all the nodes. The storage devices that allow such connectivity are

- 10 -

Sonomas and Photons. Photons need a hub to allow such direct connectivity, at least at the present time, but the nature of the data integrity problem remains the same whether there is a hub present in the storage sub-system or not. We will discuss the issues of performance and data availability of such an architecture in the next section.

Before we discuss the proposed modifications to the system software for ensuring the integrity of data, we should explore the conditions under which the SCSI-2 reservations are not adequate. There are two sets of applications that can exist on some distributed systems. One set is composed of OPS and allows simultaneous access to the same shared data block by more than a single node of the cluster. The second set allows only a single node to access a block of data at any given time. These two sets can be differentiated by the volume managers that they use. OPS must use CVM while all the other applications can use VxVM. Note that some applications, i.e., as HA-NFS or Internet-Pro, can use either of the two volume managers as CVM is indeed a super-set of VxVM. Let's first discuss the situation for VxVM based applications and then move to the OPS.

Applications that use VxVM are based on the assumption that the underlying architecture is a shared-nothing architecture. What this means is that the applications assume that there is only one "master" that can access a data item at any given time. Therefore, when that master is no longer part of the cluster the resources owned by it will be transferred to a secondary node that will become the new master. This master can continue to reserve the shared resources in a manner similar to the dual-ported architectures, however, this implies that there can only be one secondary for each resource. This in fact limits the availability of the cluster as multiple secondaries that can be used are not used. To overcome this limitation we can use the forced reserve IOCTL, but this means that the current quorum algorithm, which is also based on SCSI-2 reservations can no longer be used. There are three solutions to the problem of quorum; first, we can set aside a disk (or a controller) to be used as the quorum device and do not put any shared data on that device. Since the membership algorithm guarantees (through the use of the quorum device) that there will be only one primary component in the system, we can be sure that nodes doing the forced reservation for the shared devices are indeed the ones that should be mastering these devices in the new membership set. A second solution is to bring down the cluster when the number of failures reaches $N - 2$, where N is the number of nodes in the cluster. This obviously reduces the availability of the system, but such a cluster would

not need to use a quorum device as such devices are only used when $N = 2$. Both of these

solutions are not optimal. For the first solution we need to have a disk (for those systems

without a controller) reserved for the purpose of quorum and for the second solution we

are not utilizing the full availability of the cluster. A third, albeit more complex, solution

avoids both of these pitfalls. This solution was originally designed for a system with SDS

volume manager, but is equally applicable to a system where nodes use forced reservation.

In short, applications using VxVM can be made ultra-available (i.e., as long as there is a

path to their data they will be able to operate) and preserve the integrity of the data trusted

to them in a multi-ported architecture (such as the one shown in Figure 2) using the

currently supported SCSI-2 reservations.

      The situation for OPS is slightly more complex. OPS is based on the assumption

that each node that is running an instance of OPS is indeed capable of accessing all the data

in the database, whether that data is local or not. This assumption essentially eliminates the

use of reservations that are exclusive. While the best solution for ensuring data integrity is

to make sure that the optional SCSI-3 persistent group reservations are implemented by the

disk drive vendors, there are alternate solutions that can satisfy the data integrity

requirements. The two solutions proposed in this section are based on the observation of

the previous section; if we can ensure that the following inequality is satisfied, then we are

guaranteed that the data will remain integral for no-fault class of failures:

$$Max\{T_D\} < Min\{T_R\} \qquad\qquad\qquad (1)$$

In equation (1), T_D *is* the time it takes to detect the failure of the communication medium

and start a reconfiguration and T_R is the time it takes for the reconfiguration on a secondary

node to allow user I/O activities to commence. The current value of minimum value of T_R is

5 seconds, but that value is for a system without any application or database on it. A more

typical value for T_R is in order of (anywhere from 1 to 10) minutes. This implies that the

maximum value of T_D must be less than 5 seconds if we intend to be strictly correct. The

solutions proposed in this section are complementary. In fact, the first solution, called disk-

beat, guarantees that the inequality of equation (1) is met, and therefore, is sufficient. The

second solution is an added security measure and does not guarantee data integrity by

itself, but it can help the detection process in most cases. Therefore, the implementation of

the second solution, which relies on reconfiguration messages over the public-net, is

- 12 -

optional.

In the disk-beat solution, the Cluster Membership Monitor (CMM) creates an I/O thread that writes to a predefined location on the shared storage device its sequence number. In one embodiment, the sequence number is broadcast to other nodes through heart-beat messages (steps 302 and 304 in Figure 3). The sequence number represents, in one embodiment, the number of times the cluster to which the subject node belongs has been reconfigured. In addition, the same thread also will read the sequence number of the other nodes in the cluster (step 306). If it finds that its sequence number is lagging behind that of any other cluster members, i.e., is less than the sequence number of any other node (test step 308), then it will execute a reconfiguration (step 310). The location of the "array" of sequence numbers and their relative ordering is specified by the Cluster Data Base (CDB) file. The read and write activity is a periodic activity that takes place with period T, where $T \leq Max\{T_D\}$. Note that as the thread will be a real-time thread with the highest priority and the processor that it will be executing on does not take any interrupts, therefore, the execution of the periodic read and write as well as the commencement of the reconfiguration is guaranteed to happen within the specified time period. Further note that the disk-beat solution is merely a way of preventing two sides of a disjoint cluster from staying up for any significant period of time. Finally, note that this solution is asynchronous with respect to the reconfiguration framework and is more timely than the one used in current systems.

The second, and optional, solution relies on the public net for delivery of a message and is based on a push technique. This message is in fact a remote shell that will execute on all nodes of the cluster. Once a node hits the return transition, which may indicate that a failure has occurred, that node will do a **rsh clustm reconfig** operation on all other nodes of the cluster (step 402 in Figure 4). Since reconfiguration is a cluster wide activity and since most of the time the nodes would go through the reconfiguration in a short period of time, the addition of this call does not introduce a significant overhead in terms of unnecessary reconfigurations. As mentioned earlier, however, this solution does not guarantee the data integrity as the disk-beat solution does and will only be used in the system if it is deemed helpful for quicker detection of failures.

In any system there is a small, but non-zero, probability that the system is not making any forward progress and is "hung". In a clustered system, a node that is hung

can cause the rest of the cluster to go through a reconfiguration and elect a new membership that does not include the hung node. While the disk-beat solution of previous section guarantees that a node will detect the failure of the communication medium, it is based on the assumption that this node is active. The fencing mechanisms, such as those based on the SCSI-2 exclusive reservation schemes, can protect the data integrity in the case of nodes that are inactive, or hung, for a period of time and then come back to life and start issuing I/O commands against the shared database. To raise the level of protection against such failures for systems that cannot use the exclusive reservations, we use a second protection mechanism based on the fail-fast drivers. This scheme is already in place, implemented as part of the original membership monitor, and works by having a thread of the CMM arming and re-arming a fail-fast driver periodically. If the node cannot get back to re-arm the driver in its alloted time, the node will panic. Let's say that the periodic re-arming activity will happen with period T_{ff}. If we can guarantee the inequality of equation (2) is satisfied then we can guarantee that there will be no new I/O issued against the shared data base when the node "wakes up".

$$Max\{T_{ff}\} < Min\{T_R\}$$

(2)

It is important to note that the fail fast driver is scheduled by the clock interrupt, the highest priority activity on the entire system. However, the fail fast does not happen as part of the interrupt and there is a window during which the I/O already committed to the shared database may be written to it. Since the fail fast driver is handled by the kernel and has the priority of the kernel, this window will be small. In addition, nodes that are hung are typically brought back to life via a reboot. This, by definition, eliminates this node as a potential source of data corruption. The only situation which may allow our proposed system to compromise the integrity of the data due to a no-fault failure is a hung node that is brought back to life as a member of the cluster and has user I/O queued up in front of the two highest priority tasks in the system. Needless to say, this is an extremely unlikely scenario.

- 14 -

5 Analysis and Conclusions

Above, we have looked at the issue of data integrity for two different architectures of clustered systems. The difference in the cluster architecture is due to the differences in the storage sub-system. In one case the storage sub-system is entirely composed of dual-ported storage devices and in the other case the storage sub-system is entirely composed of multi-ported devices. We classified and analyzed the various types of failures that a clustered system can see and tolerate and discussed the methods the system employs in the current dual-ported architectures to ensure data integrity in presence of such failures.

As pointed out above for applications that utilize the VxVM volume manager, the addition of multi-ported storage sub-system does not pose any danger to the data integrity. In fact, for such systems the multi-ported storage sub-system increases the degree of high-availability to $(N - 1)$, where N is the number of nodes in the cluster. To achieve this level of high-availability we found out that some modification to the quorum algorithm is necessary.

For the applications that use CVM as their volume manager, i.e., OPS, the lack of low level SCSI-2 reservations in multi-ported architectures can be overcome with the introduction of disk-beat. This technique will protect the data integrity against the same class of failures that disk fencing does and showed an additional algorithm based on the use of public net that can further improve the timing of reconfigurations, in general, and detection of failed communication medium, in particular. While a multi-ported architecture does not suffer from a lack of guaranteed data integrity in our system, it enjoys the inherent benefits of additional availability and increased performance. Performance is enhanced on several fronts. First, no software solution is needed to create the mirage of a shared-disk architecture. Second, with the use of RAID-5 technology the mirroring can be done in hardware. Finally, such the system can truly tolerate the failure of $N - 1$ nodes, where N is the number of nodes in the cluster, and continue to provide full access to all the data.

The comparison of dual-ported and multi-ported architectures done in this paper clearly indicate the inherent advantages of the multi-ported systems over dual-ported systems. However, dual-ported systems are able to tolerate some benign malicious failures and pilot errors that are not tolerated by multi-ported systems. While these are nice features that will be incorporated into the multi-ported architectures with the implementation of SCSI-3 persistent group reservations, they only affect OPS and are not general enough to

- 15 -

guard the system against an arbitrary malicious fault or even an arbitrary inadvertent fault.

The above description is illustrative only and is not limiting. The present invention is therefore defined solely and completely by the appended claims together with their full scope of equivalents.

* *

What is claimed is:

1.   A method for detecting failure within a clustered system within a
predetermined amount of time, the method comprising the steps of:
        performing the following steps with a frequency having a period less than
    the predetermined period of time:
            sending from a subject node sequence data identifying a particular
        configuration of a cluster to which the subject node is a member;
            receiving, from each of one or more other nodes of the cluster,
        respective corresponding sequence data;
            comparing the sequence data of the subject node to the
        corresponding sequence data of each other node of the cluster; and
            detecting a failure pertaining to the subject node if the sequence
        number of the subject node identifies a configuration of the cluster which
        precedes a configuration of the cluster identified by any sequence number
        corresponding to any of the other nodes.

2.   The method of Claim 1 further comprising the step of:
        sending reconfiguration messages to the other nodes through a public
    network in response to detecting the failure.

3.   A method for detecting failure within a clustered system within a
predetermined amount of time, the method comprising:
        arming a fast-fail driver which is configured to detect failure with the
    clustered system;
        attempting to re-arm the fail-fast driver within the predetermined amount of
    time; and
        detecting failure to re-arm the fast-fail driver within the predetermined
    amount of time.

4.   The method of Claim 3 further comprising:

refraining from issuing data access requests to one or more shared devices

in response to detecting failure to re-arm the fast-fail driver within the

predetermined amount of time.

5.   The method of Claim 3 wherein the predetermined amount of time is less

than a minimum amount of time required to complete reconfiguration of the clustered

system.

6.   A computer readable medium useful in association with a computer which

includes a processor and a memory, the computer readable medium including computer

instructions which are configured to cause the computer to detect failure within a clustered

system within a predetermined amount of time by:

performing the following steps with a frequency having a period less than

the predetermined period of time:

sending from a subject node sequence data identifying a particular

configuration of a cluster to which the subject node is a member;

receiving, from each of one or more other nodes of the cluster,

respective corresponding sequence data;

comparing the sequence data of the subject node to the

corresponding sequence data of each other node of the cluster; and

detecting a failure pertaining to the subject node if the sequence

number of the subject node identifies a configuration of the cluster which

precedes a configuration of the cluster identified by any sequence number

corresponding to any of the other nodes.

7.   The computer readable medium of Claim 6 wherein the computer

instructions are configured to cause the computer to detect failure within a clustered

system within a predetermined amount of time by also:

sending reconfiguration messages to the other nodes through a public

network in response to detecting the failure.

8.   A computer readable medium useful in association with a computer which includes a processor and a memory, the computer readable medium including computer instructions which are configured to cause the computer to detect failure within a clustered system within a predetermined amount of time by:

arming a fast-fail driver which is configured to detect failure with the clustered system;

attempting to re-arm the fail-fast driver within the predetermined amount of time; and

detecting failure to re-arm the fast-fail driver within the predetermined amount of time.

9.   The computer readable medium of Claim 8 wherein the computer instructions are configured to cause the computer to detect failure within a clustered system within a predetermined amount of time by also:

refraining from issuing data access requests to one or more shared devices in response to detecting failure to re-arm the fast-fail driver within the predetermined amount of time.

10.   The computer readable medium of Claim 8 wherein the predetermined amount of time is less than a minimum amount of time required to complete reconfiguration of the clustered system.

11.   A computer system comprising:

a processor;

a memory operatively coupled to the processor; and

a failure detection module (i) which executes in the processor from the memory and (ii) which, when executed by the processor, causes the computer to detect failure within a clustered system including the computer system within a predetermined amount of time by:

performing the following steps with a frequency having a period less than the predetermined period of time:

sending sequence data identifying a particular configuration

- 19 -

of a cluster to which the computer system is a member;

receiving, from each of one or more other nodes of the

cluster, respective corresponding sequence data;

comparing the sequence data of the computer system to the

corresponding sequence data of each other node of the cluster; and

detecting a failure pertaining to the computer system if the

sequence number of the computer system identifies a configuration

of the cluster which precedes a configuration of the cluster identified

by any sequence number corresponding to any of the other nodes.

12.   The computer system of Claim 11 wherein the failure detection module is configured to cause the computer to detect failure within a clustered system within a predetermined amount of time by also:

sending reconfiguration messages to the other nodes through a public

network in response to detecting the failure.

13.   A computer system comprising:

a processor;

a memory operatively coupled to the processor; and

a failure detection module (i) which executes in the processor from the

memory and (ii) which, when executed by the processor, causes the computer to

detect failure within a clustered system within a predetermined amount of time by:

arming a fast-fail driver which is configured to detect failure with the

clustered system;

attempting to re-arm the fail-fast driver within the predetermined

amount of time; and

detecting failure to re-arm the fast-fail driver within the

predetermined amount of time.

14.   The computer system of Claim 13 wherein the failure detection module is configured to cause the computer to detect failure within a clustered system within a predetermined amount of time by also:

refraining from issuing data access requests to one or more shared devices

in response to detecting failure to re-arm the fast-fail driver within the

predetermined amount of time.

15.   The computer system of Claim 13 wherein the predetermined amount of

time is less than a minimum amount of time required to complete reconfiguration of the
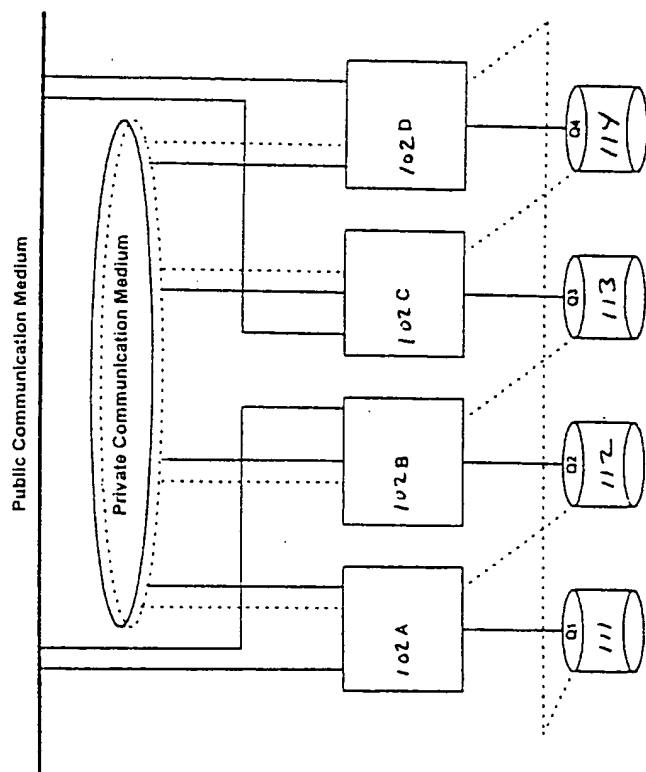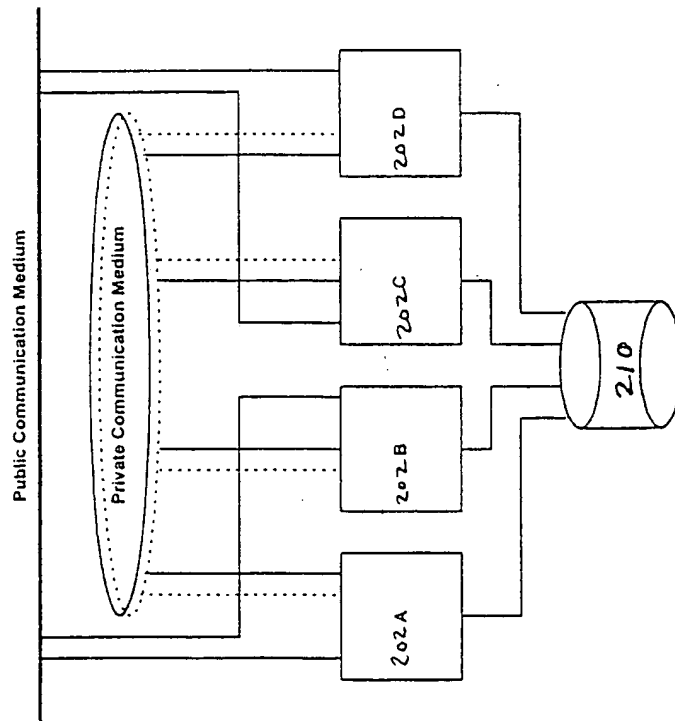
clustered system.

Figure 1

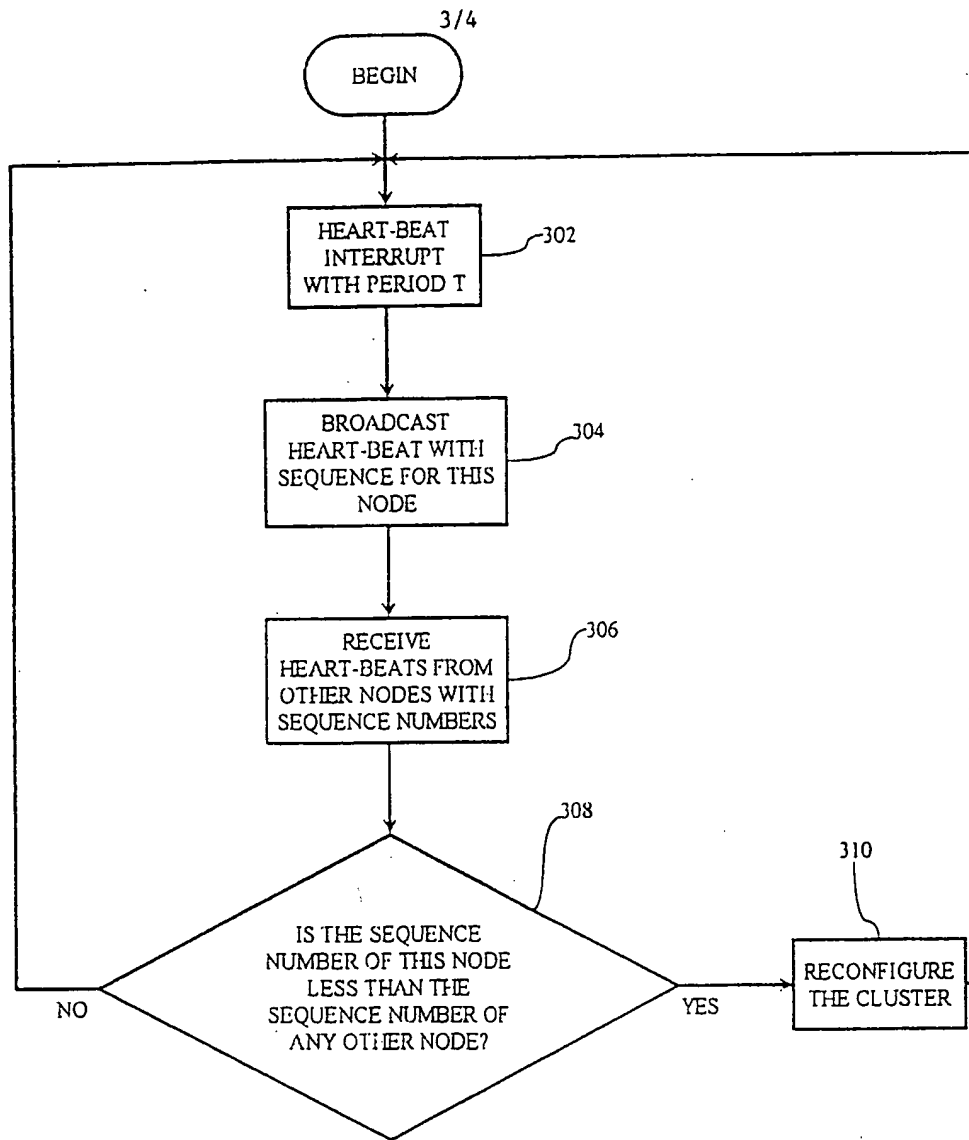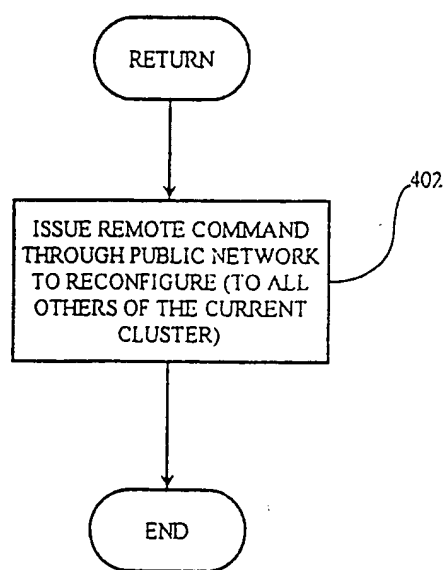Public Communication Medium

Private Communication Medium

202 A

202 B

202 C

202 D

210

Figure 2

BEGIN

HEART-BEAT
INTERRUPT
WITH PERIOD T   ⌐302

BROADCAST
HEART-BEAT WITH
SEQUENCE FOR THIS
NODE   ⌐304

RECEIVE
HEART-BEATS FROM
OTHER NODES WITH
SEQUENCE NUMBERS   ⌐306

308

IS THE SEQUENCE
NUMBER OF THIS NODE
LESS THAN THE
SEQUENCE NUMBER OF
ANY OTHER NODE?

NO

YES

310

RECONFIGURE
THE CLUSTER

FIG. 3

4/4

```
                    ╭─────────╮
                    │ RETURN  │
                    ╰────┬────╯
                         │
                         ▼
        ┌────────────────────────────┐        402
        │  ISSUE REMOTE COMMAND      │ ⟍_____
        │  THROUGH PUBLIC NETWORK    │
        │  TO RECONFIGURE (TO ALL    │
        │  OTHERS OF THE CURRENT     │
        │  CLUSTER)                  │
        └──────────────┬─────────────┘
                       │
                       ▼
                 ╭─────────╮
                 │   END   │
                 ╰─────────╯
```

FIG. 4

INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6   G06F11/14   G06F11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 6   G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 772 126 A (SUN MICROSYSTEMS INC) 7 May 1997	3,4,8,9, 13,14
A	see column 2, line 1-11	5,10,15
A	see column 5, line 37 - column 6, line 47; figure 3.	1,2,6,7, 11,12

A	US 5 390 326 A (SHAH NITIN J) 14 February 1995 see column 16, line 3-25; figure 5	1,2,6,7, 11,12

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

Date of the actual completion of the international search	Date of mailing of the international search report
12 March 1999	24/03/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Huyghe, E

1

Information on patent family members

Ir   ational Application No

PCT/US 98/22160

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0772126	A	07-05-1997	EP 0801746 A JP 9237226 A WO 9716744 A		22-10-1997 09-09-1997 09-05-1997
US 5390326	A	14-02-1995	NONE		